Where to Go from Here
Taking Stock: Where Are We?
Goals for this Course

- Learn how to model and solve complex problems with computers.
- To that end:
  - Explore common abstractions for representing problems.
  - Harness recursion and understand how to think about problems recursively.
  - Quantitatively analyze different approaches for solving problems.
What We’ve Covered

Strings
Streams
Recursive Problem-Solving
Stacks
Queues
Vectors
Maps
Sets
Lexicons
What We’ve Covered

Recursive Graphics
Recursive Enumeration
Recursive Backtracking
Big-O Notation
Sorting Algorithms
Class Design
Pointers and Memory
Constructors and Destructors
What We’ve Covered

Dynamic Arrays
Chained Hashing
Linear Probing
Robin Hood Hashing
Linked Lists
Binary Search Trees
Huffman Coding
Graphs
You didn’t just learn a list of concepts.
You learned to make those concepts *shine*.
Assignment 1: Strings, Streams, and Recursion
Assignment 2: Container Types
Assignment 3: Memoization, Recursive Optimization
Assignment 4: Recursive Backtracking
Assignment 5: Big-O, Sorting
This tool displays the strongest recent earthquakes reported by the US Geological Survey. You can use the controls on the side of the window to select the time interval you're interested in. This visualizer will show the 5 strongest earthquakes within that interval.

Remember that the earthquake magnitude scale is logarithmic. An earthquake that is one magnitude in strength higher than another releases around 32 times as much energy.

- Magnitude 7.5 115km ESE of Palora, Ecuador at 02:17:22 AM on Feb 22, 2019
- Magnitude 7 27km NNE of Azangaro, Peru at 12:50:41 AM on Mar 01, 2019
- Magnitude 6.4 116km SE of L'Esperance Rock, New Zealand at 07:46:14 AM on Mar 06, 2019
- Magnitude 6.4 49km NW of Namatanai, Papua New Guinea at 06:35:55 AM on Feb 17, 2019
- Magnitude 6.2 Northern Mid-Atlantic Ridge at 11:57:05 AM on Feb 14, 2019

**Assignment 6**: Classes, Dynamic Arrays
### Assignment 7: Hash Functions, Class Design

<table>
<thead>
<tr>
<th></th>
<th>Chained Hashing</th>
<th>Linear Probing</th>
<th>Robin Hood Hashing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert (success)</td>
<td>758.11ns</td>
<td>388.44ns</td>
<td>406.33ns</td>
</tr>
<tr>
<td>Insert (failure)</td>
<td>424.51ns</td>
<td>247.08ns</td>
<td>262.46ns</td>
</tr>
<tr>
<td>Lookup (success)</td>
<td>411.30ns</td>
<td>244.01ns</td>
<td>265.86ns</td>
</tr>
<tr>
<td>Lookup (failure)</td>
<td>346.17ns</td>
<td>250.69ns</td>
<td>237.27ns</td>
</tr>
<tr>
<td>Remove (success)</td>
<td>451.11ns</td>
<td>242.85ns</td>
<td>447.46ns</td>
</tr>
<tr>
<td>Remove (failure)</td>
<td>285.53ns</td>
<td>251.65ns</td>
<td>240.45ns</td>
</tr>
</tbody>
</table>

- \( \alpha = 0.5 \)
- \( \alpha = 0.6 \)
- \( \alpha = 0.7 \)
Assignment 8: Linked Structures
Assignment 9: Trees and Tree Searches
Computer science is more than just programming.
These skills will make you better at whatever you choose to do.
So what comes next?
What exactly is computer science?
How do I learn more about it?
The CS Core

- **Systems**
  - CS106B: Programming Abstractions
  - CS107: Computer Organization and Systems
  - CS110: Principles of Computer Systems

- **Theory**
  - CS103: Mathematical Foundations of Computing
  - CS109: Intro to Probability for Computer Scientists
  - CS161: Design and Analysis of Algorithms
The CS Core

- **CS106B**: Programming Abstractions
- **CS107**: Computer Organization and Systems
- **CS110**: Principles of Computer Systems
- **CS103**: Mathematical Foundations of Computing
- **CS109**: Intro to Probability for Computer Scientists
- **CS161**: Design and Analysis of Algorithms

**Systems**

**Theory**
The CS Core

- CS106B: Programming Abstractions
- CS103: Mathematical Foundations of Computing
- CS109: Intro to Probability for Computer Scientists
- CS161: Design and Analysis of Algorithms
- CS107: Computer Organization and Systems
- CS110: Principles of Computer Systems
How does the computer work, at its most basic levels?

How do those low-level details lead to larger-scale phenomena?

What levels of abstraction lie beneath basic C++ concepts?
How can we use software to control hardware devices?

How do displays, keyboards, etc. get data into or out of the computer?

What’s it like to build a computer system from scratch?
The CS Core

CS106B
Programming Abstractions

CS107
Computer Organization and Systems

CS110
Principles of Computer Systems

CS103
Mathematical Foundations of Computing

CS109
Intro to Probability for Computer Scientists

CS161
Design and Analysis of Algorithms
The CS Core

CS106B
Programming Abstractions

CS107
Computer Organization and Systems

CS110
Principles of Computer Systems

CS103
Mathematical Foundations of Computing

CS109
Intro to Probability for Computer Scientists

CS161
Design and Analysis of Algorithms
What mathematical tools can we use to analyze programs, processes, and graphs?

Why are some problems harder to solve than others?

Are there problems that cannot be solved by computers, and how would we know?
The CS Core

**Systems**
- **CS107**
  - Computer Organization and Systems

**Theory**
- **CS103**
  - Mathematical Foundations of Computing
- **CS109**
  - Intro to Probability for Computer Scientists
- **CS161**
  - Design and Analysis of Algorithms

**Programming Abstractions**
- **CS106B**
- **CS110**
  - Principles of Computer Systems
Why is a randomly-built binary search tree probably balanced?

How do we use computers to make sense of large data sets?

What is machine learning, and how do machines learn?
The CS Core

- **CS106B**: Programming Abstractions
- **CS107**: Computer Organization and Systems
- **CS103**: Mathematical Foundations of Computing
- **CS109**: Intro to Probability for Computer Scientists
- **CS110**: Principles of Computer Systems
- **CS161**: Design and Analysis of Algorithms
Next Steps in CS

- It’s reasonable to take one of CS107, CS103, or CS109 as a next CS class. You’ll put in a good amount of work and learn a ton in the process.

- *Do not feel pressured to do everything at once.* Taking two of these classes concurrently is a significant amount of work, and it isn’t expected of you.

- Want some more guidance? Come talk to me after class!
Want to explore a bit?
**CS5xx**
*(e.g. CS547, CS520, CS522, CS523, etc.)*

- Any CS class of the form CS5xx is a seminar course.
- Frequently (not always), it’s “show up, listen to interesting people speak, and take away what you will.”
- These can be great ways to see what different fields are like, and the speakers are often really inspiring.
CS300  
Departmental Lecture Series

- CS300 is a course where CS professors present about their research work.
- It was originally intended for PhD students; it’s now open to everyone.
- Only offered fall quarter; highly recommended if you want to learn more about what’s out there.
**CS106E**

*Practical Exploration of Computing*

- Broad survey of computing topics, including
  - how the internet works,
  - computer security,
  - how operating systems work,
  - bits and bytes, and
  - web programming.

- Great course if you’re interested in working in the software industry in a non-technical capacity.
Want to learn specific technologies?
Any class of the form CS193x is an introduction to a particular language or technology. For example:

- CS193A: Android Programming
- CS193C: Client-Side Web Technologies
- CS193I: iOS Programming
- CS193P: iPhone and iPad Programming
- CS193Q: Accelerated Intro to Python

Great for learning specific tools.
Explore what C++ programming looks like outside of CS106B.

Get exposure to the standard libraries and some really, really cool techniques beyond what we saw here.

Excellent next step if you’d like to work in C++ going forward.
Want to dive deeper?
What values does our technology reflect?
How can we make sense of contemporary political and social debates over technology?
What are the historical precedents for what we’re seeing play out now?

This class is truly fantastic. Please take it!
CS147
Intro to Human-Computer Interaction

• How does design thinking apply in computer science?
• How do you prototype, evaluate, and refine a new piece of software?
• Prerequisite: CS106B! ✓
The CS Major
Thinking about CS?

• Good reasons to think about doing CS:
  • I like the courses and what I’m doing in them.
  • I like the people I’m working with.
  • I like the impact of what I’m doing – or I want to steer how technology is developed and used in the world.

• Bad reasons to think about not doing CS:
  • I really enjoy this, but other people are better coders than me.
  • I’m learning a lot, but other people have been doing this longer than me and there’s no way for me to catch up.
  • I like the classes I’m taking, but the field is so big and I have no idea which area to focus in.
  • I don’t know what I’m going to be doing many years down the line, and I don’t want to be pigeonholed into just a tech person.
The CS Major

• A common timetable:
  • Aim to complete *most* of the core by the end of your sophomore year (probably CS106B, CS103, CS107, CS109, and one of CS110 and CS161).
  • Explore different tracks in your junior year and see which one you like the most.
  • Spend your senior year completing it.
• It’s okay if you start late!
  • The latest time you can *comfortably* start a CS major would be to take CS106A in winter quarter of sophomore year.
  • And the coterm is always an option!
For more information, visit

https://csmajor.stanford.edu/
The CS Coterm
What’s the Coterm?

• It’s a *coterminal master’s degree*.

• Work concurrently on your BS (in any subject) and your MS (in computer science).

• Designed with two populations in mind:
  • Give existing CS majors access to more depth and breadth of knowledge.
  • Give non-CS majors a chance to explore CS and emerge with a thorough command of the material.

• All Stanford undergrads are welcome to apply. This is intentional, and the door is open to all comers!
What’s the Coterm Like?

- The MS is 45 units, including the CS core (renamed the “foundations requirement”), breadth, depth, and electives.
- Expect to take three CS classes per quarter once you’re at “grad status.”
- TA (teaching) and RA (research) positions are available to offset tuition.
  - Full-time (“50%”) TA/RA positions offset tuition and provide a stipend.
  - Part-time (“25%”) TA/RA positions offset some tuition and provides a (smaller) stipend.
- With a 50% TA/RA position, you’re limited to 10 units/quarter. But that’s probably a good idea anyway.
Why Coterm?

- TA and RA positions are available to offset the cost.
  - Some of my best TAs did their undergrad in comparative literature, anthropology, and physics.
  - This is a great way to deepen your understanding of the material.
- Thinking about applying?
  - *Take enough CS classes to establish a track record.*
    - CS106B on its own is probably not sufficient. Completing most of the CS core probably is.
  - *Maintain a solid CS GPA.*
    - You don’t need a 4.0 in your CS classes. You should aim to do well and make sure you’re enjoying what you’re doing.
- The department loves having cotermals around. We don’t have a quota, and (empirically, based on no insider knowledge) the application process is designed to accept as many qualified people as possible.
For more information, visit

https://cs.stanford.edu/admissions/current-stanford-students/coterminal-program
The CS Minor
What’s the CS Minor?

- Five classes in CS: take CS103, CS107, CS109, plus two other depth classes.
- Nice option if you want to keep exploring CS while pursuing another major.
- For more information, visit [https://cs.stanford.edu/degrees/ug/Minor.shtml](https://cs.stanford.edu/degrees/ug/Minor.shtml)
Outside Stanford
Learning More

• Some cool directions to explore:
  • **Specific technologies**. You already know how to program. You just need to learn new technologies, frameworks, etc.
  • **Algorithms**. Learn more about what problems we know how to solve.
  • **Software engineering**. Crafting big software systems is an art.
  • **Machine learning**. If no new ML discoveries were made in the next ten years, we’d still see a huge impact.
How to Explore Them

- Online courses through Coursera, Udacity, edX, etc. are fantastic ways to learn new concepts.
  - Andrew Ng’s machine learning course, Fei Fei Li’s computer vision course, Tim Roughgarden’s algorithms course, and Jennifer Widom’s databases courses are legendary.
- Learning by doing is the best way to pick up new languages and frameworks.
  - Find a good tutorial (ask around), plan to make a bunch of mistakes, and have fun!
- Know where to ask for help.
  - Online resources like Stack Overflow can provide help (if you know how to ask questions well; that can take some practice!)
Some Words of Thanks
Who's Here Today?

- Aero/Astro
- African / Afro-American Studies
- Bioengineering
- Biology
- Business
- Chemical Engineering
- Chemistry
- Civil and Environmental Engineering
- Classics
- Computer Science
- Creative Writing
- Earth Systems
- Economics
- Education
- Electrical Engineering
- Energy Resource Engineering
- English
- Environmental Systems Engineering
- Environment and Resources
- Ethics in Society
- Geophysics
- Human Biology
- Immunology
- Individually-Designed
- International Policy
- International Relations
- Law
- Linguistics
- Management Science and Engineering
- Materials Science and Engineering
- Mathematical and Computational Science
- Mathematics
- Mechanical Engineering
- Music
- Physics
- Psychology
- Public Policy
- Science, Technology, and Society
- Sociology
- Symbolic Systems
- Theater and Performing Studies
- Undeclared!
My Email Address

htiek@cs.stanford.edu
You now have a wide array of tools you can use to solve a huge number of problems.

You have the skills to compare and contrast those solutions.

You have expressive mental models for teasing apart those problems.
My Questions to You:

What problems will you choose to solve?
Why do those problems matter to you?
And how are you going to solve them?